Problem H

Simplified λ-evaluations

source: `lambda.*`

Lambda calculus is the main theoretical core of functional languages. It is based on evaluating λ-expressions.

A λ-expression is a string of characters consisting either of *1)* a single constant (for example t), or *2)* a function definition written as $L\langle var\rangle.\langle body\rangle$, where L denotes λ, $\langle var\rangle$ is always some variable (for example x) that can occur in the $\langle body\rangle$, which is again some λ-expression, or *3)* an application of some $\langle function\rangle$ (for example f) to an $\langle argument\rangle$ (for example x), in our case written as $(f)x$, both $\langle function\rangle$ and $\langle argument\rangle$ are λ-expressions.

The occurences of variable $\langle var1\rangle$ in an expression E are called bound, if they are inside of the $\langle body\rangle$ of some sub-expression $L\langle var1\rangle.\langle body\rangle$ of E. Variable occurences that are not bound are called unbound.

Evaluation of λ-expression results in another λ-expression and is performed using the following rules:

1. Constant is evaluated to itself,

2. Function definition evaluates to itself,

3. Function application $(\langle function\rangle)\langle argument\rangle$ is evaluated as follows: first, the $\langle function\rangle$ is evaluated to $L\langle var\rangle.\langle body\rangle$ or something else. In the latter case, the $\langle argument\rangle$ is evaluated, and the whole function application evaluates to $(\langle evaluated\ function\rangle)\langle evaluated\ argument\rangle$. If the $\langle function\rangle$ evaluates to $L\langle var\rangle.\langle body\rangle$, the $\langle argument\rangle$ is not evaluated, but all unbound occurences of variable $\langle var\rangle$ inside of the expression $\langle body\rangle$ are directly replaced by (substituted with) the argument $\langle argument\rangle$. The result of the whole evaluation of function application is then the evaluated $\langle body\rangle$, after the substitution was performed.

We limit the expressions to contain a single-character lowercase ($a \ldots z$) variables and constants only. Sometimes the evaluation never stops. Your program should perform at most 1000 function applications when evaluating any λ-expression. During all evaluation steps, the evaluated λ-expression will not exceed 10000 characters.
Here are some examples of evaluation of simple λ-expressions (shown in several steps, the text behind ; is a comment):

```
1.
  (Lx.x)y                     ; x <- y
  y

2.
  ((Lx.Ly.(x)y)Lz.z)Lq.q      ; x <- Lz.z
  (Ly.(Lz.z)y)Lq.q            ; y <- Lq.q
  (Lz.z)Lq.q                  ; z <- Lq.q
  Lq.q
```

Note that the scope of the variable next to λ covers only the body of the lambda expression, and the same variable can occur at other places of the expression, for example:

```
3.
  ((Lx.x)(Ly.y)Lx.x)x
  ((Ly.y)Lx.x)x
  (Lx.x)x
  x
```

The body of the λ-expression can contain unbound variables - constants:

```
4.
  (((Lx.Ly.q)Lz.t)r)u    ; x <- Lz.t
  ((Ly.q)r)u             ; y <- r
  (q)u
```

Finally note that our evaluation is simplified as compared to the real λ-calculus: when the argument is substituted for all unbound occurences of some variable during function application, some of the variable occurences in the argument that were previously unbound can become bound... This is normally solved by renaming all the variables in the argument before the function application, and keeping track of correct substitutions. In this problem, you don't need to take care of this.

```
5.
  ((Ly.Lx.y)x)w           ; y <- x
  (Lx.x)w
  w
```

The Department of Programming Languages decided to implement a new functional language. However, they first need a core engine that will evaluate λ-expressions. Write a program that will read a set of expressions from the input file and output their evaluations.

Input specifications

The input contains set of λ-expressions, one per line. Only the last line contains the expression consisting of a single constant z. You can assume that the input is correct.

Output specifications

The output should contain the result of evaluation of the expressions on the input in the same order as they appear in the input including the last line. If the expression leads to more than 1000 function applications, the line should contain single word "`unterminated`".

Sample input

```
Lq.q
((Lx.Ly.(x)y)Lz.z)Lq.q
(Lx.x)x
((((Lm.Ln.Lf.Lx.((m)f)((n)f)x)Lo.Lt.(o)t)Lu.Lv.(u)(u)v)a)b
(Lx.(x)x)Lx.(x)x
(q)(Lx.Lx.x)z
z
```

Output for sample input

```
Lq.q
Lq.q
x
(a)(a)(a)b
unterminated
(q)Lx.x
z
```